# **Introduction**

## **1.1. Objective**

This project is trying to use YOLO to detect several different targets among 15000 images including biker, car, pedestrian, traffic Light, traffic Light-Green, traffic Light-Green Left, traffic Light-Red, traffic Light-Red Left, traffic Light-Yellow, traffic Light-Yellow Left, and truck in images.

Those objects are classified and localized by the YOLO algorithm with a probability to which a certain object belongs and its corresponding bounding box. YOLOv3 is used in this project because it is fast, accurate, robust, and fully developed.

15000 images with 512\*512 pixels from Udacity self-driving car dataset in roboflow is used for a pre-trained model which is trained on COCO datasets.

## **1.2. Significance**

Autonomous driving will be the leading way of transportation in the future, because it lowers the risk of making mistakes, increases stability and safety, and provides passive protection beyond human cognition. In the meanwhile, an automatic system with dedicatedly constructed planning strategies improves the efficiency of road planning, decision making, and so on and so forth, liberating drivers while increasing comfortability. The current autonomous driving systems rely highly on visual information to make decisions, e.g., detecting multiple objects simultaneously, estimating the depth map of an image, semantically segmenting multiple instances. Another important aspect of autonomous driving is to make decisions in real-time, as the lower the latency is, the safer the system will be. Therefore, recognizing multiple targets in real-time is highly demanded in the decision-making of self-driving systems.

This leads us to the YOLO (you only look once) algorithm which has the capability of detecting multiple objects in real-time with reasonable performance. The Original YOLO paper demonstrates its ability on performing object detection on videos with 45 FPS, and even 150 FPS with a tiny version. At that time, the other state-of-the-art methods, e.g., Faster R-CNN is only able to process images at 7 FPS. The essential idea of YOLO is that it provides a unified neural network to train and do inference instead of a multi-stage system inferring on a neural network several times. The motivation to do so is that feeding data to a (convolutional) neural network is an extremely time-consuming process even on GPU. The R-CNN-based methods are either running a neural network multiple times during inference, or running multiple neural networks to do regional proposals and detection, separately. The inherent computation of selecting regional proposals is the key factor that R-CNN based method can not run inference in a fast matter. The millisecond-level inference enables YOLO to be way better than human cognitions whose reaction time is always at half-second.

Furthermore, not only is YOLO fast, it is also accurate in terms of the mean average precision (mAP). The original paper demonstrates its mAP is doubled compared to the other state-of-the-art work. For autonomous driving, precision is of equal importance as real-time compatibility. The higher accuracy the algorithm is able to achieve, the less risk the driver will face. For instance, it is disastrous for an autonomous driving system to fail to detect a pedestrian, a car in front of a driver. YOLO makes big progress in real-time detection. It is the most popular and reliable way to detect multiple targets in real-time based on the image information obtained from the camera, that is also the reason why we choose this system as a main approach to the detection in the project. A combination of YOLO, Lidar, Radar, and HD(high-definition map) leads to a reliable, safe auto-driving system that will bring huge pleasure and convenience for humen.

In this project, the main methodology behind the YOLO will be reviewed in detail. In the meanwhile, some previous and current state-of-the-art work will also be reviewed in a brief manner, which may lead us to the motivation of YOLO. Finally, the experiments on a custom dataset were performed to demonstrate the performance of the YOLO algorithm in terms of mean average precision and visual inspection.

# **2. Methodology**

## **2.1. Current State of Research**

Object detection is one of the most significant and hottest high-level vision tasks which empowers a variety of applications in autonomous driving, surveillance, machine inspection, and etc. The fundamental problem of object detection is to classify potential objects within an image while localizing them simultaneously. The former task categorizes the objects within an image or a patch of an image, if patch-based object detection methods were considered. In general, classification is not an overwhelmingly intimating task, the convolutional neural network (CNN) revolutionizes the accuracy of object classification year by year, e.g., ImageNet[7], AlexNet[7], ResNet[8], etc. On the contrary, localization is comparatively harder to achieve even in a fully supervised manner. There are three main reasons that contribute to that: 1) the deformation of a variety of objects, 2) the (partial) occlusion of objects under a variety of physical condition alterations; 3) the appearance of the same objects in different scales.

Even before the emergence of convolutional neural networks, there were many state-of-the-art methods that always used hand-crafted features (e.g., spatial representation, local statistics) to represent an object. The Haar-based rectangular features were firstly used to detect faces, which characterizes the representation of the face by finding that the intensities of the chuck are different from those of the other regions, and areas between eyes are darker than other areas. By taking advantage of that observation, Viola et al[17], proposed using the gradient of the intensity in neighboring regions along with a cascaded AdaBoost classifier to detect the face. Later on, some features based on local statistics are used to detect people or some other objects. Dalal & Triggs[18] proposed using histograms of oriented gradients (HoG) features for human detection based on the local statistics of a human object. Felzenszwalb et al[19]., proposed using the deformable representation of the object as the feature to detect a variety of objects not stick to a certain one. Their hypothesis is that an object always has a hierarchical representation, e.g, the face contains eyes, mouth, nose which each of them is also a single object which can be identified.

However, the hand-crafted features sometimes are biased in some cases, especially when an occlusion exists. Ideally, features ought to be learned from the data itself. Therefore, we only focus on CNN-based methods in this paper. But notice that Transformer[9] and Mixer[10] are also other powerful architectures other than CNN, which imposes different mechanisms to represent spatial features.

At the very early stage of the research in object detection, people always constructed a two-stage system: the first is to train a classifier based on sub-patches of images where a certain object is labeled; the second is to perform inference with the trained classifier at different scale (from coarse to fine) and aggregate the results at different scales as the final localization, because objects are always at different scales. One of the most widely-used methods is sliding windows at different scales, which sweeps overlapping patches of an image with a set of scaled windows and performs inference at each window. This method is computationally expensive in reality especially in the cases where the classifier is (convolutional) neural network. Another issue of this method is the ill-posedness of precise localization on deformable objects. The Overfeat method proposed by Sermanet[4] et al mitigates the inefficiency in the computation of the sliding window by applying the sliding window directly over the feature map of the CNN instead of over the input images. The key idea is that convolution can preserve spatial information. Performing a sliding window over the input image and feeding the CNN is equivalent to feeding the whole image to the CNN and performing a sliding window over the feature map which can be done by another convolutional layer. This method, however, still needs to perform coarse-to-fine inference which means running the network several times. Girshick[11] et al., mitigate this issue by proposing the Regional proposal method named R-CNN. The essential idea is to reduce the number of potential patches which contain objects by performing a preliminary segmentation over the raw image. By doing so, only the patches (~2k) potentially contain the objects that will be feeding to the neural network to extract the feature and then train a support vector machine (SVM) to classify the image based on the extracted features. Even though this method reduces the potential patches containing objects to the network, it runs a network for each patch which is time-consuming. The fast R-CNN[12] and faster R-CNN[13] aim to mitigate this issue by changing the way to do regional proposals. The fast R-CNN feeds the whole image in a CNN and performs regional proposals over the feature map meaning only one network is running for each image. The faster R-CNN proposed a regional proposal network (RPN) to do the regional proposal which essentially takes advantage of the fact that the extracted feature used for regional classification can also be used to do a regional proposal.

YOLO (You Only Look Once) is a powerful new technology in computer vision. It only applied one single neural network to the full image which improved the speed and brought the objects-detection to real-time. As its name suggests, it can identify multiple specific objects in videos, live feeds, or images by only “looking” at the images at once. There are some differences and similarities between YOLO and the two localization ways: sliding window slides all the image or subsets regions in the image. Deformable parts models (DPM) is one of the methods using sliding window. R-CNN uses region proposals instead of sliding window, and then use selective search to find potential bounding boxes, which is very time-consuming and has a high requirement of precision for each stage. Different from R-CNN, Fast and Faster R-CNN uses neural network instead of selective search which improve the speed and precision. However, it still cannot be real-time. There are some research

## **2.2. The development of YOLO**

YOLOv1 is a powerful objective detection method that was proposed by R. Joseph[1] in 2015. Different with deformable parts models (DPM) using sliding windows slides the entire image, and R-CNN using region proposal method to generate all potential bounding boxes and classify on the objects. YOLOv1 only includes a simple convolutional network that simultaneously predicts multiple bounding boxes and their class probabilities[1]. That is also the reason why it is fast and can be used as real-time objective detection. The main disadvantage of YOLOv1 is that its fast identify speed leads to it underperforming other detection methods in accuracy.

RCNN-based methods firstly select the bounding boxes potentially contains objects and then regress bounding boxes coordinates via Convolutional neural network and classify each bounding box with SVM. Through the Faster RCNN uses a regional neural network to extract regional proposals, it is still computationally expensive regarding real-time object detection. YOLO gets rid of the regional proposals by dividing an entire image into **SxS** grids and for each grid there are **B** bounding boxes. Essentially, YOLO assumes there might be some objects within a small grid of the image, if so bounding boxes should be applied to localize those objects. In the meanwhile, the YOLO makes strong assumptions: 1) there are only one type of object within a single grid; 2) the bounding boxes within a grid should not be too large. Under this assumption we further derive the formulation of object detection in YOLO. Firstly, we assign a confidence score to each bounding box as , where IOU is the intersection over union, which is a method to measure the accuracy of detections. The equation of IOU is: . The confidence score will be the IOU between the predicted box and the ground truth box[1], which means the area of overlap divided by the area of union between the predicted box and the ground truth box. There are 5 parameters to help to locate the object,  and the confidence of each object.  is the center of the bounding box,  is the width and is the height of the bounding box. Secondly, for each grid, as the assumption suggested, only a probability of a single object will be predicted, considering there are C types of objects, we formally derive the predict probability is . Finally, we multiple these two probabilities to predict the probability whether there is an object exists within a grid, the probability that the object belongs to which class.

Under the above derivation, YOLO essentially views object detection as a regression problem- regress the corrdinates of the bounding boxes and their corresponding probabilities. This can be implemented as a Convolutional neural network. Essentially, the YOLO learns each bounding box through the label instead of selecting the regional proposal as what RCNN-based methods do. On the contrast, dividing an image into SxS grids is trivial to implement. A few tricks need to be metioned to get fully understanding of the YOLO, the first is that convolutional operator is spatial-variant. By taking advantage of that, convolution over an entire image will give us all the results of convolution over each bounding box. Essentially, an entire image is fed into a CNN to extract the feature map. By doing so, all features maps of all bounding boxes can be generated in a single pass, but the question is that we do not know which feature is associate with which bounding box. The solution to that is to take advantage of the label to help us learn the bounding box directly. The idea is that we know the number of grids and bounding boxes and their corresponding location coordinates. From the extracted feature map, we essentially regress them with fully connected layer to directly output the location coordinates and label probabilities of each bounding box. By doing so, we do not need to care about which feature in the feature map corresponds to which bounding box, essentially we learn them through the labels via a fully connected layer. Because the final purpose is to regress each bounding boxes with their extracted feature map. In the real implementation, the fully connected layer can be implemented convolutionally to compute the predictions over all the bounding boxes instead of one by one, as informed by [4]. During inference, non-maximal suppression is further applied to aggregate the predicted bounding boxes to generate the final predictions in case there are multiple highly overlapped bounding boxes around a single object.

The architecture of YOLOv1 is shown in Figure 1 as below:

Diagram

Description automatically generated

**Figure 1**: The architecture of YOLOv1

As it is shown in Figure 1, there are 24 convolutional layers and 2 fully connected layers.

In the training process, there are multiple bounding boxes in each grid. The bounding box with the biggest IOU is the one to do the prediction. The unified loss is optimized by a specially constructed squared error which includes: the confidence score loss, the bounding box coordinate predictions loss, and classification loss. The equation of unified loss is shown below:

Since no object cell will push to its confidence score as zero, which will overpower the gradient of cell contains object. and are two parameters to balance it. means j-th bounding box in cell I is the one to do the prediction. means if cell I contain object.

The limitation of YOLOv1 is that it has strong spatial constraints on the bounding box: there are only two bounding boxes in one cell and only can have one class. That is also the reason YOLOv1 struggles with small objects among groups. The other limitation of YOLOv1 is that it cannot deal with deformed objects. The last limitation is that the loss function treats the error the same no matter it is in a small or large bounding box. However, a small error in a small bounding box will lead to a worse IOU than a small error in a big bounding box. That limitation cause the main source of error in the prediction.

YOLOv2 and YOLO9000 were proposed by R. Joseph[22] in 2017 which can detect more than 9000 categories with a high speed in real-time. The model size of YOLOv2 can be changed to trade off the accuracy and speed. The reseason why YOLOv2 can detect so many different objects is that it proposed a method to use the data they already have and then extend it to its detection system. WordTree is used in YOLOv2. It is a hierarchical directed graph structure that combines different sources of data so that it riches the number of detection categories. In addition, YOLOv2 used a joint training algorithm, which used the labeled images to localize objects and meanwhile used classification images to make the model robust and stable. The main improvement compared to YOLOv1 is that batch normalization was added to all convolutional layers. The other improvement is YOLOv2 used a high-resolution classifier that improve the accuracy. YOLOv2 used anchor boxes to predict bounding boxes instead of the fully connected layer which is in YOLOv1. Although the accuracy is slightly decreased but improves the recall rate. K-means is used to cluster box dimensions. From the result of testing, when k = 5, there is a good trade-off between high recall and model size. The input of image size is changed every few iterations, so that it can do the prediction among different resolutions which makes the tradeoff between speed and accuracy easier in YOLO9000.

YOLOv3 was proposed in 2018[23]. The main difference with YOLOv2 is it used Darknet-53, which has 53 convolutional layers, to extract features. It makes YOLOv3 faster than YOLOv2. Better than YOLOv1 and YOLOv2, YOLOv3 has good performance with AP 13.3 on detecting small objects. However, YOLOv3 has slightly worse performance on detecting medium and large size objects than small grouped objects. In addition, YOLOv3 makes it easier to do the trade-off between speed and accuracy which just needs to change the model size. YOLOv3 is a fully developed, fast, and accurate real-time objects detection system. In mAP(mean average precision) measured at 0.5 IOU, YOLOv3 is on par with Focal Loss but about 4x faster[6]. Which inspired by ResNet and FPN (Feature-Pyramid Network) architectures, YOLO-V3 uses a feature extractor called Darknet-53 (it has 52 convolutions) that contains skip connections (like ResNet) and 3 prediction heads (like FPN) — each processing the image at a different spatial compression[3].

## **2.3. Data Collection[21]**

The dataset is a public dataset from roboflow, called Udacity Self Driving Car Dataset. It contains 15000 images with 11 classes including biker, car, pedestrian, traffic Light, traffic Light-Green, traffic Light-Green Left, traffic Light-Red, traffic Light-Red Left, traffic Light-Yellow, traffic Light-Yellow Left, and truck in images. 97942 labels across those images, 1720 images with no labels which means there are no objects in it. Some examples of images without label are shown below:

A picture containing text, sky, outdoor, tree

Description automatically generated

**Figure 2**: examples of images without label

## **2.4. Model Architecture**

The architecture of YOLOv3 is shown in figure 3 as below[26]:

Text, whiteboard

Description automatically generated

**Figure 3**

Different with YOLOv2, YOLOv3 uses 53 layers convolutional networks. More 53 layers were added, because it is a detection task. Darknet-53 was used to extract features.

YOLOv3 codes are downloaded from[20], which is an official implementation in PyTorch. The version of python needs to be above 3.8.0 and the version of PyTorch needs to be above 1.7. I used the YOLOv3 package to do the multiple objects detection by using labeled images. There is no change in the code to adapt the dataset. YOLOv3 system can train on the custom dataset. There are some requirements for the custom dataset: images per class bigger than 1500, instances per class bigger than 10000, representative, each image need to be labeled, the images that contain no object should within 10% of total datasets. I used Udacity Self Driving Car Dataset as my train dataset. Firstly, I clone the repo and install all required packages. Then, the pretrained model, YOLOv3, Large, was chose. Since the training of YOLO is computationally expensive, a pre-trained model which is trained in a big dataset with multiple targets. The pretrained model was pretrained on COCO128. The pretrained weight is attached to the homework folder. 80% of the 15000 images are used to train the model, 10% are used in the validation test. The last 10% of the dataset are used as testing.

The image size is 512\*512 in the input. Since the pretrained model was used and COCO dataset included cars, bikes, traffic lights, etc., there is no need for big training epochs. 50 epochs were used in the training. I use the biggest batch size 16 that my computer allows. The other hyperparameters are default in the training.

The evaluation of predicted bounding boxes is intersection over union (IOU) metric. The equation of IOU is shown in the development of YOLO. Generally, the IOU score is set with a constant threshold of 0.5. A predicted bounding box is considered correct if its IOU scores bigger than 0.5 in this project.

# **3. Results and Discussion**

The result of training and validation loss are shown in figure 4:

Graphical user interface

Description automatically generated

**Figure 4**. training, and validation loss

Box loss is:

Object loss is:

CLS loss is:

The precision and recall of the model also are shown in figure 4. Mean average precision(mAP) is used as the evaluation criterion. Figure 4 showns the result of when threshold of mAP = 0.5, and the average from mAP = 0.5 TO mAP = 0.95. The detail of those result is attached to the homework folder.

The recall vs confidence graph is shown in figure 5, the precision vs confidence curve is shown in figure 6, the recall vs precision graph is shown in figure 7 and the confusion matrix of the prediction is shown in figure 8 as below:

Chart

Description automatically generated

**Figure 5**. recall vs confidence curve

Chart

Description automatically generated

**Figure 6**. precision vs confidence curve

Chart

Description automatically generated

**Figure 7**. precision vs recall curve

Chart, waterfall chart

Description automatically generated

**Figure 8**. confusion matrix

There are two files as the input: images and labels. Figure 9 is what the images look like after images are labeled.

A picture containing text, different, various, several

Description automatically generated

**Figure 9**. batch 0 in training

The comparation of the predicted images by YOLOv3 and the real labeled images are shown in figure 10 to figure 15.

A picture containing outdoor, crowd

Description automatically generated

**Figure 10**. real labeled images of batch 0

A picture containing text, outdoor, crowd

Description automatically generated

**Figure 11**. predicted images of batch 0

A picture containing outdoor, light, different, way

Description automatically generated

**Figure 12**. real labeled images of batch 1

A picture containing outdoor, light

Description automatically generated

**Figure 13**. predicted images of batch 1

A picture containing outdoor, city, different, day

Description automatically generated

**Figure 14**. real labeled images of batch 2

A picture containing outdoor, city, different, day

Description automatically generated

**Figure 15**. predicted images of batch 2

From figure 10 to figure 14, we can see even targets are not clear to tell, hidden by trees, tiny in the images, and grouped shows, YOLOv3 still has very good performance.

This project is a practice of using the YOLO system to do multiple objective detections. There is not much innovation in this work. Since I and my advisor work on small datasets and are still in the step of developing our own algorithm, there is not a very good application by using machine learning methods for my own research till now. However, I am sure we will use machine learning to improve our algorithm in the future. The reason why I chose YOLO as the topic of my final project is that I used to work for a technology company that is focused on autonomous driving during the summer after I got my master’s degree. I had no idea about machine learning and worked in a quality engineering position at that time, and felt very curious about how they deal with it. This report concludes with knowledge from published papers. It is only used to learn and practice YOLO, and will not be used for publishing.

# **References**

[1] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[2] <https://pjreddie.com/darknet/yolo/>

[3] <https://towardsdatascience.com/yolo-v3-explained-ff5b850390f>.

[4] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229.

[5] <https://medium.com/@anand_sonawane/yolo3-a-huge-improvement-2bc4e6fc44c5>

[6] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[7] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3), 211-252.

[8] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[9] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

[10] Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., ... & Dosovitskiy, A. (2021). Mlp-mixer: An all-mlp architecture for vision. arXiv preprint arXiv:2105.01601.

[11] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[12] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[13] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28, 91-99.

[14] <https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/>

[15] <https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52>

[16] <https://sandipanweb.wordpress.com/2018/03/11/autonomous-driving-car-detection-with-yolo-in-python>

[17] Viola, P., & Jones, M. (2001, December). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001 (Vol. 1, pp. I-I). Ieee.

[18] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05) (Vol. 1, pp. 886-893). Ieee.

[19] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2009). Object detection with discriminatively trained part-based models. IEEE transactions on pattern analysis and machine intelligence, 32(9), 1627-1645.

[20] <https://github.com/ultralytics/yolov3/wiki/Train-Custom-Data>.

[21] <https://public.roboflow.com/object-detection/self-driving-car>.

[22] Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).

[23] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[24] <https://viso.ai/deep-learning/yolov3-overview/>

[25] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[26] <https://dev.to/afrozchakure/all-you-need-to-know-about-yolo-v3-you-only-look-once-e4m>